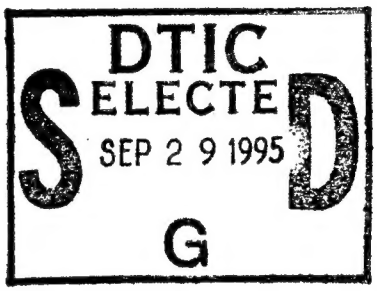
RL-TR-95-100
Final Technical Report
June 1995

HANDLING CRITICAL SYSTEM REQUIREMENTS IN ADAPTIVE SYSTEMS

SRI International

Li Gong and Teresa Lunt



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-95-100 has been reviewed and is approved for publication.

APPROVED: Emilie J. Siarkiewicz

EMILIE J. SIARKIEWICZ
Project Engineer

FOR THE COMMANDER: John A. Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3AB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1995	Final Apr 94 - Oct 94

4. TITLE AND SUBTITLE
HANDLING CRITICAL SYSTEM REQUIREMENTS IN ADAPTIVE SYSTEMS

5. FUNDING NUMBERS
C - F30602-94-C-0092
PE - 33140F
PR - 7820
TA - 04
WU - PA

6. AUTHOR(S)
Li Gong and Teresa Lunt

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
SRI International
Computer Science Laboratory
333 Ravenswood Ave
Menlo Park CA 94025-3493

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3AB)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-95-100

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Emilie J. Siarkiewicz/C3AB/(315) 330-2135

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
Computer systems implementing mission-critical applications typically must be capable of adapting to and responding to events and changes in the operational environment. To accomplish this, a system often will have to make dynamic tradeoffs in meeting its various and often conflicting objectives: for example, security and availability. In this final report, we describe a general framework within which multiple critical system properties can be specified, multiple requirements can be jointly considered, and conflicts can be resolved. We give examples to illustrate how the framework can be useful, and suggest future work on the application of the framework.

14. SUBJECT TERMS
Critical System Requirements, Computer Security, Fault Tolerance, Security Policy, Adaptive Systems, Availability, Real Time, Formal Specification

15. NUMBER OF PAGES
40

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	U/L

Contents

1 Introduction

For many mission-critical applications, different modes of operation are applicable in different circumstances. Therefore, systems implementing such applications must be capable of adapting to and responding to events and changes in the operational environment. To accomplish this, a system often must make dynamic trade-offs in meeting its various objectives, including security, integrity, timeliness, availability, fault tolerance, and safety. For instance, after evaluating a situation and determining that the mission benefit outweighed the security risk, a commander might transmit sensitive information in the clear when no secure communication line is available.

To achieve multiple critical system objectives with a high degree of assurance, each of these objectives must be considered in the overall system security policy, so that they are considered in a unified context and framework, and trade-offs can be made when various requirements conflict. In previous work, such as the Secure Alpha project [2], the system design explicitly recognizes the potential for conflicts and trade-offs between two system objectives: namely, real-time and security.

However, such a system usually considers a very small number (usually two) of conflicting requirements. Whenever another requirement becomes a significant factor, the whole model and system policy have to be reworked. In addition, such a system tends to consider only the specific requirements (and trade-offs) present in that system, with the consequence that the results cannot be easily extended to include additional requirements or to apply to other systems. For example, the control of trade-offs is often "hardwired" into system code so that any modification requires a significant amount of effort.

This final report summarizes our effort in addressing the above issues. Specifically, we propose a more general framework within which multiple critical system properties can be jointly considered. The framework includes components for the following functions: specifying multiple requirements, checking the feasibility of the requirements, assigning priorities, and resolving conflicts, and consists of five major steps: (1) establishing policy goals, (2) establishing system constraints, (3) deciding feasibility of goals, (4) providing directions for trade-offs, and (5) realizing trade-offs. Our approach is generic because the trade-offs and the rules will typically be application specific. We use examples, with specific conflict resolution rules, to illustrate how the framework can be useful.

Our framework can be utilized in a number of ways, the most significant being the following.

- Construct a tool for system analysis and simulation so that
  - Potential conflicts and trade-offs can be discovered at design time
  - Trade-off strategies can be tested by simulation or analysis
  - Important design decisions can be made with these conflicts in consideration.

- Implement a generic controlling module that can realize many types of security policies and can be plugged into existing and future systems.

The rest of this report is organized as follows. We first give an abstract description of our framework and describe the specification techniques that we often use in the framework. After that, we study three examples to demonstrate the use of the framework. We conclude with a summary and suggestions for future work.

2 Project Objective

As we argued in the previous section, systems currently under development tend to pay little attention to adaptivity and its implications for security – in particular, the necessary trade-offs between security and various other critical system properties. Moreover, the investigation into potential conflicts in system requirements is usually very limited in scope, and the mechanisms to deal with such conflicts are usually difficult to extend to handle additional requirements or to generalize to different systems. Therefore, this project's objective is to propose a general framework within which multiple requirements can be expressed in a uniform fashion and the trade-offs between them can be studied with systematic methods.

Recall that an adaptive system may experience many different modes of operation and may be constrained by unexpected environmental factors. Therefore, a general framework in which to consider the conflicts and trade-offs between various critical system requirements must be

- *Extensible*, so that adding another requirement is easy
- *Generic*, so that adding another *type* of requirement is easy
- *Flexible*, so that the framework can express a wide range of system requirements
- *Heterogeneous*, so that the requirements can be qualitative and/or quantitative
- *Tractable*, so that requirements can be evaluated systematically and efficiently.

Within this framework, a requirement can specify that a hard deadline must be met, and can be expressed in quantitative terms. A requirement can be "soft" or "hard". For example, the network delay for a particular environment is fixed, and is thus a non-negotiable constraint. The fault tolerance level of an operational environment is on the other hand a soft constraint, i.e., a negotiable one, because it is possible to change (weaken) this requirement if the given system configuration cannot meet it.

Constraints and requirements can be qualitative as well as quantitative because sometimes we cannot express a qualitative rule (for making trade-offs) in quantitative terms. For example, there might be a rule that when security and availability are in conflict, availability takes precedence over security. Such a rule may be captured in the evaluation and decision procedure, but may or may not be recast accurately as a quantitative requirement.

To summarize, the framework, as illustrated in Figure 1, must enable us to translate specifications, constraints, and objectives (or goals) – for example, of systems, environment, and applications – into mathematical formulas, so that we can use systematic methods to examine them, to detect conflicts, and to evaluate the feasibility of trade-offs.

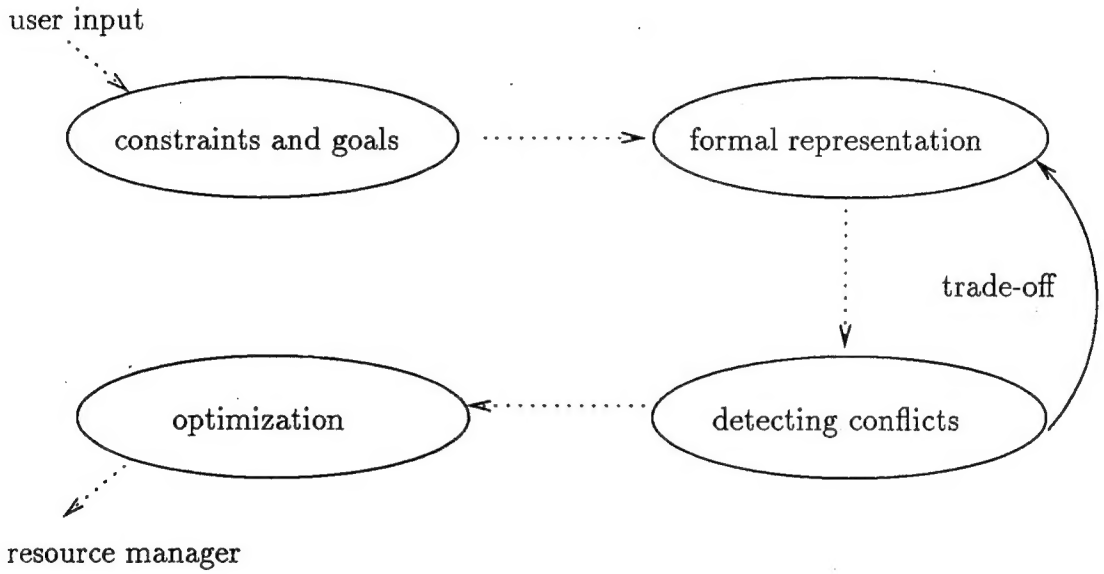


Figure 1: A General Framework

3 Framework Description and Usage

Our framework, which is the process of analyzing the target system and of obtaining a specification of the system security policy, involves five important steps.

Step 1 (Establishing Policy Goals) *Understand and write down system requirements in precise, mathematical and logical terms.*

For example, it is insufficient to require that a file system to respond "fast". A more precise statement is that "a read access is serviced within 30ms", or `time_to_read` $\leq$ `30ms`. If the

real-time requirement may differ from 30ms, the bound should be replaced by a variable, which is substituted for each instance of the application.

Another example is a limit – an upper bound – on the covert-channel bandwidth. For a particular application, the requirement may be `bandwidth` $\leq$ `0.001bps` (bits per second).

Step 2 (Establishing System Constraints) *Understand and write down the target system characteristics that are directly or indirectly related to and can affect the realization of the policy goals.*

For example, the completion speed of a network protocol involving multiple hosts (and hence hops) may depend on the actual number of hosts involved. In other words, a possible constraint is that `protocol_finish_time` $\geq$ `a + b` $\times$ `n`, where a and b are constants. This constraint may affect the realization of the policy goals because of another constraint: `task_completion_time = protocol_finish_time + ....`

Step 3 (Deciding Feasibility of Goals) *Use mathematical methods to seek a feasible solution of policy goals under the given system constraints.*

Given a precise definition of the policy goals and the system constraints, we conform and extend them to a set of mathematical equations and use standard methods to decide if there is any solution. (More details in the next section.) A solution implies that the multiple policy goals can in fact be satisfied under the current system constraints. A verdict of no solution, on the other hand, indicates that to complete the task, either some policy goals must be weakened or the system constraints must be altered by changing system configuration (e.g., adding hardware to improve performance).

Step 4 (Providing Directions for Trade-offs) *Specify rules and priorities of trade-offs between potentially conflicting policy goals and system configurations.*

When some policy goals conflict with each other or are incompatible with (i.e., unrealizable in) the current system, we must identify ways to resolve the conflict. In the former case, it is desirable to specify which goal should be sacrificed first, and by how much. For example, if `time_to_read` $\leq$ `30ms` cannot be satisfied (e.g., due to conflicting covert channel bandwidth constraint), we can choose to weaken it to `time_to_read` $\leq$ `60ms`, instead of relaxing security requirement. In the latter case, if the reason of conflict is slow hardware, we can improve the system construction to maintain the same goal as before, i.e., `time_to_read` $\leq$ `30ms`. Such trade-offs rules and priorities must be specified in suitable mathematical forms for automated manipulation. More details are given in the next section.

Step 5 (Realizing Trade-offs) *Use mathematical methods to seek a feasible and perhaps the best solution of policy goals under the given system constraints, following the trade-off rules.*

Given a set of policy goals, a set of system constraints (some of which can be improved), and a set of trade-off rules, we can use mathematical tools, such as linear programming, to obtain (the best) trade-offs between critical system requirements.

Table 1 summarize the major steps of our framework. In the next section, we provide more details on the methods and techniques of specifications.

Step 1	Establishing Policy Goals
Step 2	Establishing System Constraints
Step 3	Deciding Feasibility of Goals
Step 4	Providing Directions for Trade-offs
Step 5	Realizing Trade-offs

Table 1: Major steps of the framework

4 Expressing Policy Goals and System Constraints

Our approach is to model the system requirements with m equations:

$$\begin{cases} a_{11} \times x_1 + a_{12} \times x_2 + \ldots + a_{1n} \times x_n = b_1 \\ a_{21} \times x_1 + a_{22} \times x_2 + \ldots + a_{2n} \times x_n = b_2 \\ \qquad\qquad \ldots \\ a_{m1} \times x_1 + a_{m2} \times x_2 + \ldots + a_{mn} \times x_n = b_m \\ x_i \geq 0, \quad i = 1, 2, \ldots, n \end{cases}$$

Sometimes we represent this set of equations simply as $A \cdot x = b$. We need to determine only whether the set of equations has feasible solutions. If so, the requirements can be satisfied. If not, the requirements contain conflicts that must be resolved.

Although x_i's are restricted to non-negative values, and we write equations and not inequalities, it is easy to see that these restrictions do not limit the scope of the application [21]. For example, given a requirement of the form

$$\begin{cases} a \times x = b \\ x \text{ is unrestricted} \end{cases}$$

we can introduce two new variables x' and x'' and rewrite the above as the following requirements:

$$\begin{cases} a \times (x' - x'') = b \\ x', x'' \geq 0 \end{cases}$$

Similarly, a requirement of the form

$$\begin{cases} a \times x \leq b \\ c \times y \geq d \end{cases}$$

can be rewritten as follows:

$$\begin{cases} a \times x + x' = b \\ x' \geq 0 \\ c \times y + y' = d \\ y' \leq 0 \end{cases}$$

When we further restrict some variables to non-negative integer values, we can capture a wider range of requirements. For example, to express the following logical relationship between the values of two variables,

$$\textbf{if } x > a \textbf{ then } y \geq b$$

we can write an equivalent set of requirements as follows:

$$\begin{cases} x \leq z \times a \\ y \geq (1 - z) \times b \\ 1 \geq z \geq 0, \quad z \text{ being integer variable} \end{cases}$$

Similarly, to express the relationship:

$$\textbf{either } x = a \textbf{ or } x = b$$

We can write equivalently as follows:

$$\begin{cases} x = z \times a + (1 - z) \times b \\ 1 \geq z \geq 0, \quad z \text{ being integer variable} \end{cases}$$

If the requirements do not exhibit conflicts, there may be more than one feasible solution. In this case, one solution may be more attractive than another, based on a certain cost criteria, say a "cost" function. Suppose that the smaller the cost, the better the solution, then we can reformulate our requirement and add a new goal to minimize the cost, thus forming a linear programming problem [22, 21].

$$\begin{cases} \texttt{minimize cost(x)} \\ \hline A \cdot x = b \\ x \geq 0 \\ A \text{ being a matrix and } x \text{ being a vector} \end{cases}$$

Note that maximization can be expressed with minimization, since maximizing $cost(x)$ is equivalent to minimizing $-cost(x)$. An example of the optimization problem is the following.

$$\begin{cases} \texttt{minimize } x_1 + 2x_2 + 5x_3 \\ \hline x_1 + x_2 + x_3 = 9 \\ x_1 - x_2 \leq 5 \\ x_1, x_2, x_3 \geq 0 \end{cases}$$

Sometimes, such an optimization technique can be used to capture trade-off rules. For example, suppose that there is a rule that security takes priority over performance, which means that the execution time (denoted as `exec_time`) for a task need not be strict if security is at risk, but the smaller the delay, the better. We can encode this flexibility of performance requirement as follows. Suppose that b is the original deadline. Then we can replace `exec_time` $\leq$ `b` with

$$\begin{cases} \texttt{minimize } z \\ \hline \texttt{exec\_time} \leq b + z \\ z \geq 0 \end{cases}$$

The above problem formulation allows the execution time to be greater than b, but also strives to minimize any delay beyond time b. Suppose that there is also an upper limit (b') on how slow the execution can be, we can add another constraint of the form $z \leq b' - b$, as follows.

$$\begin{cases} \texttt{minimize } z \\ \hline \texttt{exec\_time} \leq b + z \\ z \leq b' - b \\ z \geq 0 \end{cases}$$

Suppose that another rule states that availability takes priority over security. This means that the security level t is to some extent flexible. Suppose that f is the minimum security level. Then we can replace a constraint $t \geq f$ with

$$\begin{cases} \texttt{maximize}\, w \\ t = f + w \\ w \geq 0 \end{cases}$$

Moreover, rules can be assigned with priorities, so that when a conflict exists, certain rules are exercised first; and if conflicts still cannot be resolved, then more rules are activated. Suppose that we need to express the fact that it is desirable that minimizing z has priority over maximizing w. Given a sufficiently large positive number M, we can reflect the above priority in the following requirement:

$$\begin{cases} \texttt{minimize}\, (M \times z - w) \\ \texttt{exec\_time} \leq b + z \\ t = f + w \\ z \geq 0 \\ w \geq 0 \end{cases}$$

Because of M, the optimization tends to increase w as much as possible first, while keeping z to the minimum.

To summarize, given that the requirements can be expressed as a set of equations, we can detect conflicts in the requirements by attempting to solve these equations. If there is a solution, then there is no conflict. Otherwise, trade-offs are necessary to resolve the conflicts. Rules for resolving conflicts can sometimes be interpreted as optimization goals.

There exists a rich body of work on determining whether a set of equations has feasible solutions. For example, when the equations are linear equations over real variables, we can use the standard Gaussian elimination method to check for any conflict between the requirements. When the equations are linear equations over integer variables, the integer programming problem is known to be NP-complete. Here, the method by Shostak [25, 26] can be used, among others. Some of these methods are also available as part of the well-known theorem-proving system PVS [20].

When there is conflict, trade-off rules encoded into requirements can take effect. For solving the general optimization problem, many linear (and possibly integer) programming techniques exist [12, 21, 22]. The best known is perhaps the Simplex method of Dantzig [6]. In this method, it is possible to set up the algorithm so that one variable is optimized while another is fixed – a useful way to implement trade-off rules with priorities. The worst-case complexity of the Simplex method is exponential; but in most cases, as extensive experience

in this field shows, the Simplex method is remarkably effective even for large problems. Although the problem of linear programming has been proven to be in P [8, p.339], the most effective algorithms seem to be variations of the Simplex method [12, 21, 22].

5 Case Study I: Security and Real Time

A real-time system attempts to manage its resources so as to satisfy the timing constraints imposed by an application, while security requires the separation and encapsulation of resources for access control. Given a set of tasks with predetermined interactions, and assuming that the system does not overload, these tasks – typically having different security levels – can have their timing constraints satisfied and at the same time be prevented from interfering with one another (thus creating covert channels). The Secure Alpha project [2] addresses such applications, which are inherently multilevel in nature but are also required to provide timely responses, even in emergencies and overload situations. These exceptional situations cannot be avoided, because the system must be able to respond to an unpredictable environment. These situations, however, can bring about covert channels, as illustrated in Figure 2, where two jobs competing for resources are submitted at the same time for scheduling.

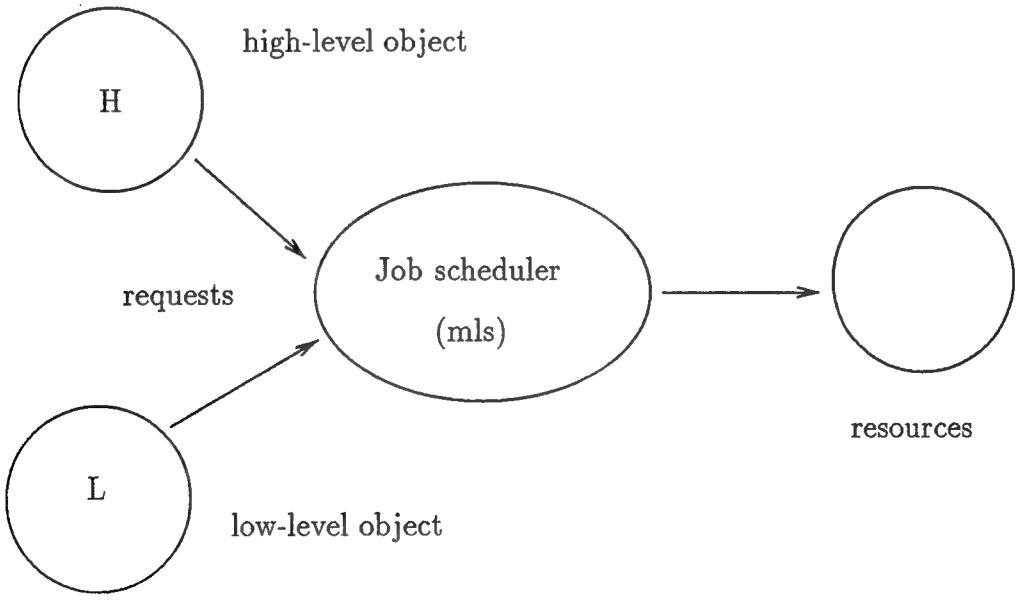


Figure 2: Security and Real Time

Because of limited resources, the scheduler cannot allocate resources to both jobs at the same time. If the high-level job is not allocated resources and is delayed, its timing constraint

may not be met. On the other hand, if the low-level job is delayed, then the low-level job can learn of the existence of the high-level job, and thus, a covert channel exists. In fact, all typical real-time scheduling policies – such as static priority, earliest deadline first, and best effort – can be exploited to covertly pass information across security levels [2]. Therefore, in step 1 of our framework, we can identify the policy goals to be maximizing its return in quick system response while keeping the covert-channel bandwidth low.

Step 1:

Policy Goal 1	`maximize time-value-function f(schedule)`
Policy Goal 2	`bandwidth` $\leq$ `limit`

Note that Secure Alpha uses a best-effort scheduling policy, which associates a value with a single computation that is a function $f()$ of the schedule, which determines when the computation completes. Therefore, the overall system goal is to maximize this time-value function while maintaining the limit on the covert-channel bandwidth.

In step 2, the internal knowledge of Alpha tells us that the covert-channel bandwidth depends on the particular schedule of tasks. In other words, for a suitable function $g()$:

Step 2:

System Constraint	`bandwidth = g(schedule)`

Theoretical examination easily shows that it is not always possible to meet the policy goals under the system constraint (step 3). In addition, actually experiment suggested that this is the case even with quite modest goals and constraints [2].

Therefore, there must be trade-offs between security and real time, e.g., between covert-channel bandwidth and real-time constraints. Secure Alpha employs the following solution. When a mission can meet all timeliness requirements, then security risk can be limited by restricting the bandwidth of the covert channel. If the mission is jeopardized, for example, in part by covert-channel countermeasures, the countermeasures can be relaxed in a well-defined and controlled way to decrease the risk of overall mission failure, as follows. The system dynamically measures the covert-channel bandwidth to assess the system security risk, with an upper limit on the bandwidth according to the security policy. This arrangement allows the system to adapt to the current situation (but within limits), in that a low bandwidth indicates that the system would be able to tolerate an additional signal, while a high bandwidth indicates that the system would need a compelling reason to accept more security risk.

In other words, the priority is given to maintaining the limit on covert-channel bandwidth, and the real-time response goal can be sacrificed. In our framework, this trade-off can be formulated as follows.

Step 4:

$$\begin{cases} \texttt{maximize time} - \texttt{value} - \texttt{function f(schedule)} \\ \hline \texttt{bandwidth} = \texttt{g(schedule)} \\ \texttt{bandwidth} \leq \texttt{limit} \end{cases}$$

Here, this (linear programming) problem is solved whenever a new schedule is calculated. A schedule that cannot satisfy all the constraints will be discarded. By the nature of linear programming, condition `bandwidth` $\leq$ `limit` is maintained, and `time-value-function f(schedule)` is maximized under this system constraint.

In the above trade-off, `bandwidth` $\leq$ `limit` is fixed. It is possible to relax this constraint and introduce another trade-off factor by allowing the `bandwidth` to be flexible within a range. In other words, a revised policy, as formulated below, is not merely to keep the covert-channel bandwidth within the allowable limit, but also to attempt to keep it to a minimum.

Step 4 (new):

$$\begin{cases} \texttt{maximize time} - \texttt{value} - \texttt{function f(schedule)} \\ \texttt{minimize z} \\ \hline \texttt{bandwidth} = \texttt{g(schedule)} \\ \texttt{bandwidth} \leq \texttt{limit} + \texttt{z} \\ \texttt{z} \leq \texttt{range} \end{cases}$$

Here, the absolute upper-bound on the covert-channel bandwidth is no longer `limit` but is `limit+range`. However, an additional policy goal, `minimize z`, is introduced to minimize the bandwidth.

In Secure Alpha, the trade-off policy as formulated in step 4 is encoded in a resolution module [2]. This process corresponds to step 5 (the last step) in our framework.

6 Case Study II: Security and Availability

In this case study, we first motivate the trade-offs between security and availability in network authentication. We then review distribution authentication protocols and apply our framework to show how the trade-offs can be handled.

6.1 Motivation

Authentication is a process by which one party (e.g., a user, a computer) establishes its claim of identity to another party. In mutual authentication, pairs of parties satisfy each other about their identities. Typically, an authentication server provides authentication service via an authentication protocol.

An authentication service is fundamental in maintaining security in a distributed system, because identification is essential to any and all enforcement of any security policy, as well as administration activities such as accounting and audit. This service is therefore a security bottleneck; once it is compromised, no security can be guaranteed. In an open environment, an individual server may not be completely and permanently trustworthy. A benign server could fail or make mistakes; a compromised one could behave maliciously by leaking client keys or by deliberately sending bogus messages. Moreover, a centralized service could not support activities such as internal auditing, for no one would be in a position to guard the guards.

The authentication service is naturally also a performance bottleneck, because many activities cannot proceed unless the identities of those involved can be satisfactorily established. For example, authentication must be available whenever a user starts a new session, or executes a protocol such as `rlogin`, `ftp`, or `telnet`. The service is also used quite often by system utilities. One machine may need to connect to a network server to retrieve electronic mail messages every five minutes throughout the day. A failure of the authentication server would be very disruptive.

A desirable authentication service, therefore, should be highly available and highly secure at the same time. There are a number of ways to increase the availability of the service. A common approach is to replicate the authentication server so that any one of several servers can perform authentication. However, this approach reduces the level of security in that if one server is compromised, security is compromised. Another way to increase availability is to reduce dependency on the service. For example, a server could issue a certificate that is valid for a period of time, during which there is no need to further contact the server. However, this scheme also degrades the level of security in that a certificate is hard to revoke, once it is issued. The longer a certificate can remain valid, the better it is for service availability, but the worse for security. As we will discuss later, replication is also useful in dealing with untrustworthy servers to increase the security of the authentication service. However, simple replications also suggest a trade-off between increasing availability and increasing security.

A solution has been proposed, called a *distributed authentication protocol* [10], in which the authentication server is replicated in such a way that multiple servers share the responsibility of providing the authentication service and a minority of compromised servers cannot compromise the service through malicious behavior and collusion. The protocol is similar to the Needham-Schroeder authentication protocol [17], but is different in that a set of servers

provide a distributed authentication service, with each server providing only a fraction of the authentication. The protocol has the following properties: fewer than a threshold number of colluding servers cannot compromise security, e.g., by leaking information about the session keys; service is available if a threshold number of uncompromised servers are operational. Adjusting the thresholds can explore the trade-off between availability and security.

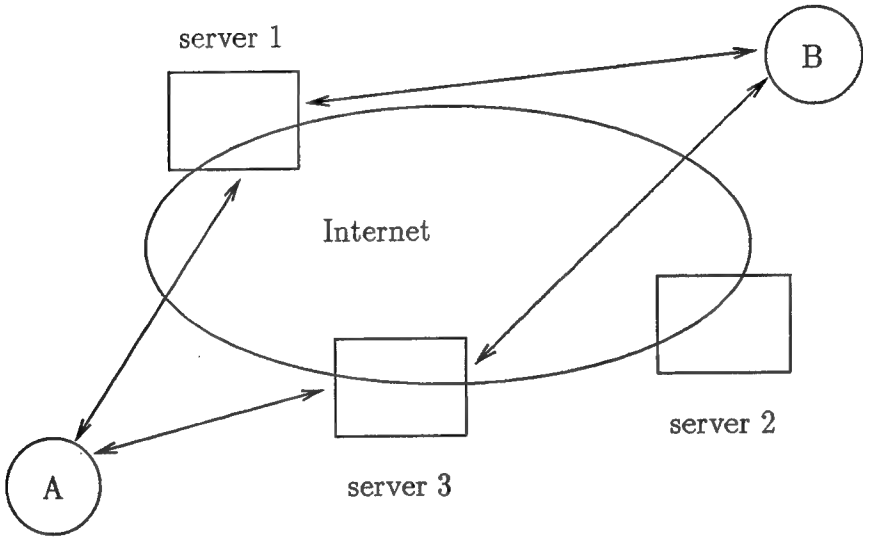


Figure 3: Security and Availability

As shown in Figure 3, two clients A and B attempt to authenticate each other using some of the n authentication servers located throughout the network. The objective is to maintain sufficient security and sufficient availability while minimizing the cost of executing the protocol.

6.2 Background on Distributed Authentication

We first examine a protocol that is similar to the Needham-Schroeder protocol [17, 18], with messages packaged in the style of Otway-Rees [19]. Client A (or B) exclusively shares a secret key Ka (or Kb) with the trusted authentication server S. By executing the following protocol, A and B intend to establish a session key Kab. We use $A \rightarrow B : m$ to denote that A sends message m to B. Let $\{m\}_k$ denote m encrypted with key k, and $(m1, m2)$ denote concatenation. A typical centralized mutual authentication protocol is as follows.

1. $A \rightarrow B : A, B, na$
2. $B \rightarrow S : A, B, na, nb$

3. $S \rightarrow B : \{B, na, Kab\}_{Ka}, \{A, nb, Kab\}_{Kb}$
4. $B \rightarrow A : \{B, na, Kab\}_{Ka}, \{na\}_{Kab}, nb$
5. $A \rightarrow B : \{nb\}_{Kab}$

In message 1 (or 2), A (or B) generates a random number na (or nb) as a challenge, known as a nonce. S generates a session key Kab and distributes it to A and B in message 3. S also encloses the nonces in the reply so that A and B can verify that the reply is fresh. Finally, A and B complete a handshake to inform each other that the correct session key has been received.

Next, we recall a *distributed authentication protocol.* Suppose n servers, $S_1, \ldots, S_n$, collectively provide an authentication service. We assume that a client (say A) registers a different key Kai with server S_i. Client A can derive these keys from a master key Ka. For example, if $h()$ is a suitable one-way hash function [7, 16], then A could register key $Kai = h(Ka, S_i)$ with server S_i. This arrangement gives extra security since one server (say S_1) does not know A's other keys $(Ka, Ka2, \ldots, Kan)$.

To prevent information leakage, a session key must not be generated by or known to a single server. In fact, there is a major difficulty in letting servers involved in choosing the key. Because A and B do not have a secure communication channel for verification purposes before authentication completes, clients cannot easily reach an agreement on what they have received from which servers. Therefore, we let both clients participate in choosing a session key; one may not trust the other's competency in this aspect. Although must be capable of generating good random numbers, a client concerned about key quality can require a few candidate keys generated by the servers or other sources and select one or exclusive-or some of them. The exclusive-or of all candidate keys will be a good key as long as at least one candidate key is good (presumably chosen by a uncorrupted server) even if some candidate keys are suspect. A preliminary version of the new protocol is as follows.

1. $A \rightarrow S_i$: A, B
2. $S_i \rightarrow A$: nsi
3. $A \rightarrow B$: $A, B, na, nsi, \{A, B, nsi, x_i\}_{kai}$
4. $B \rightarrow S_i$: $A, B, na, nb, \{A, B, nsi, x_i\}_{kai}, \{B, A, nsi, y_i\}_{kbi}$
5. $S_i \rightarrow B$: $\{B, na, y_i\}_{kai}, \{A, nb, x_i\}_{kbi}$
6. $B \rightarrow A$: $\{B, na, y_i\}_{kai}, \{na\}_{Kab}, nb$
7. $A \rightarrow B$: $\{nb\}_{Kab}$

Each participant generates a nonce (nsi for S_i, na for A, and nb for B), which is later included in encrypted messages (3 though 7) addressed to the participant so that the freshness of the messages can be established [18]. With messages 1 and 2, A obtains a nonce from each server. A chooses a candidate session key x and computes $f_{t,n}(x, i)$ for each server S_i. Here $f_{t,n}()$ is a threshold function [13] that produces n shadows of x in such a way that it is easy

to recover x from any t shadows, but less than t shadows reveals no information about x. We will explain this function later. In message 3, A sends one shadow to each server. In fact, A sends to B the shadows which are later forwarded to the servers. Similarly, in message 4, B selects y, computes the $f_{t,n}(y,i)$'s, and sends one shadow to each server. In messages 5 and 6, S_i essentially acts as a broker for A and B to exchange shadows x_i and y_i. After messages 5 and 6, B computes x from the x_i's and A computes y from the y_i's. Then they both compute session key $Kab = g(x,y)$, where $g()$ is a pre-determined one-way hash function, and complete a two way handshake.

To see how $f_{t,n}()$ works, we describe Shamir's secret-sharing scheme [24], which is based on polynomial interpolation. To compute $f_{t,n}(x,i)$, A chooses a random $(t-1)$ degree polynomial $p(x)$ with $p(0) = x$. A then computes $x_i = f_{t,n}(x,i) = p(i)$, $i = 1,\ldots,n$. Due to the property of interpolation, given any t of the x_i's, B can easily determine $p()$ and recover $x = p(0)$. With less than t shadows, no information about x can be determined. The time complexity to compute n shadows is $O(nt)$. The time complexity to recover x is $O(t \log_2 t)$. Using such a threshold scheme effectively prevents fewer than t compromised and colluding servers from leaking information about the session key, because they cannot gather enough shadows to recover x or y. In fact, they have absolutely no information about Kab.

The use of $g()$ ensures that as long as one of x and y is carefully chosen (e.g., random, never used before), Kab is likely to be a good key. More subtly, $g()$ prevents A or B from forcing a session key. To illustrate, suppose that $g()$ is replaced with an exclusive-or operation. After message 5, B knows x, and can reconstruct and retransmit another message 4 in which y' replaces y. B can choose y' to be the exclusive-or of x and the session key B wants. Similarly, A could intercept message 5 and send a new message 4 with x' to force a session key A wants. The use of $g()$ prohibits such activities. A or B can still select a session key through exhaustive search. To make this selection more difficult, the computation of $g()$ can include other information such as the participants' identifiers and the date. These considerations are important when a number of participants make collective decisions.

The execution of the protocol is aborted if an expected message does not arrive within a time-out period. An exception to this rule is message 5 (or 6). There, B (or A) waits for no longer than the time-out period. If B (or A) can recover x (or y) from the messages already received, B (or A) proceeds. Otherwise, B (or A) considers the authentication service unavailable. The length of the waiting period could be set to suit a particular environment.

A and B cannot detect illegitimate messages, however. For instance, suppose $n = (2t-1)$, then $(t-1)$ colluding servers cannot learn any information about x or y. Nevertheless, when $(t-1)$ servers send bogus shadows or faulty messages, every set of t shadows B (or A) receives may recover a different x (or y) value. In this case, A (or B) has no way to verify the legitimacy of the shadows, i.e., to determine whether the shadows received are indeed what B (or A) sent. Again this is because that before authentication completes, A and B do not have a separate secure channel to verify the legitimacy of the shadows. In fact,

they may never have such a channel if the service is bogus. In theory, A and B could try handshakes with all possible session keys derived from all combinations of the shadows until a dialogue is successful. This may not be secure in that a wrong combination may result in a cryptographically weak key that could be explored by malicious servers. Also this may not be practical if the number of combinations is large. Moreover, it is more economical if A and B can determine Kab without exchanging many messages. Therefore, for verification purposes, more redundancy of the shadows must be provided together with the shadows themselves. Shadows give A (or B) enough information to retrieve y (or x). Extra redundancy of the shadows tells A (or B) *how* to retrieve y (or x). This is in fact an authentication of the servers' behaviors, which would be impossible in a centralized approach.

The requirement of the additional redundancy is closely related to the concept of verifiable secret sharing, which is a scheme for some parties to securely share a secret by keeping different shadows, yet it is possible to verify that a shadow is legitimate. Some proposed schemes (e.g., [5]) are not very suited to the application here because they use many rounds of messages and usually require all participants' cooperation to complete the protocol. We now introduce a cross-checksum scheme as a suitable alternative. Informally, a cross-checksum scheme supplies checksums together with messages in a manner that it is possible to verify the authenticity of the messages by cross checking the checksums. We define cross checksums for x and y as $cc(x) = (g(x_1), \ldots, g(x_n))$ and $cc(y) = (g(y_1), \ldots, g(y_n))$, respectively, where $g()$ is a one-way hash function. By replacing x_i and y_i with $(x_i, cc(x))$ and $(y_i, cc(y))$ respectively in the preliminary version of the protocol, we obtain:

A Distributed Mutual Authentication Protocol.

1. $A \rightarrow S_i$: A, B
2. $S_i \rightarrow A$: nsi
3. $A \rightarrow B$: $A, B, na, nsi, \{A, B, nsi, x_i, cc(x)\}_{kai}$
4. $B \rightarrow S_i$: $A, B, na, nb, \{A, B, nsi, x_i, cc(x)\}_{kai}, \{B, A, nsi, y_i, cc(y)\}_{kbi}$
5. $S_i \rightarrow B$: $\{B, na, y_i, cc(y)\}_{kai}, \{A, nb, x_i, cc(x)\}_{kbi}$
6. $B \rightarrow A$: $\{B, na, y_i, cc(y)\}_{kai}, \{na\}_{Kab}, nb$
7. $A \rightarrow B$: $\{nb\}_{Kab}$

Adding the cross checksums does not degrade security because $g()$ is a one-way hash function (e.g., [16]). We require that given a number of pairs z's and $g(z)$'s, it is computationally infeasible to compute k from $g(k, x)$ and x. Moreover, because of the birthday paradox, if $g()$ is used in a sufficiently large amount of messages, malicious servers would be able to find x_i's that match the cross checksum $cc(x)$ by looking up a dictionary of past messages. To defeat this birthday attack, the life time of $g()$ must be limited according to the properties of the particular function. Requirement for $h()$ is similar. In fact, $h()$ and $g()$ can be the same function.

Since a legitimate message contains legitimate shadows and legitimate cross checksums, and a good (i.e., uncompromised and not faulty) server sends only legitimate messages, A and B can efficiently identify legitimate shadows, provided that more than half of the messages received are legitimate. The algorithm is defined as follows. For every $g(x_j)$ in cross checksum $cc(x)$ received from S_i, if x_j is also received from S_j, B recomputes $g(x_j)$ from x_j and compares it with the received $g(x_j)$. If the two are identical, x_j is given one credit point, or rather, S_i gives S_j one point. After all such checks are done, legitimate shadows are those from the servers which have the highest credit points, if more than half of the servers are good. To prove this algorithm correct, observe that server S_j gets a credit point from a good server S_i if and only if S_j's shadow is legitimate. Since good servers give credits to good servers and they outnumber the bad ones, servers which send legitimate messages will receive more credit points. This algorithm computes $O(n^2)$ one-way hash functions and comparisons.

In the protocol, after message 5 (or 6), B (or A) first identifies the legitimate shadows with which to recover x (or y), and then computes $g(x, y)$ to obtain the session key. B (or A) considers the service unavailable if x (or y) cannot be recovered, i.e., when less than t servers get more than t credit points. Let uc, fs, nfs, c denote, respectively, the numbers of servers that are uncompromised and operational, failed and fail-stop, failed but not fail-stop, and compromised. Then $uc + fs + nfs + c = n$. The above proof shows that the service is available if $uc > max(t-1, nfs+c)$, where typically t could be set to $t = nfs+c+1$. When shadows and checksums in faulty messages can be assumed to be random and unknown to the colluding servers, these messages are unlikely to increase the credit points of servers which send illegitimate messages. In this case, the service is available if $uc > max(t-1, c)$. When $c < t$, malicious servers may cause denial of service but cannot compromise security.

When a small number of servers are compromised, A and B may detect message losses or illegitimate messages. They could report such observations to the appropriate authorities so that suitable actions, such as inspecting suspect sites, repairing failed servers, and cleaning up compromised ones, can be taken. A client may need to change password when the accumulated total number of compromised servers since last password change is close of the threshold t. If the service is unavailable, A and B may have to try again.

6.3 Applying Our Framework

To apply our framework, let us use the following notation:

n – number of authentication servers
r – number of rounds for authentication
d – message delivery time
t – security level of distributed authentication
sec_req – security requirement of the environment
b – real-time-requirement

exec_time – actual time to complete authentication

Using this notation, we can characterize the authentication system as follows. (This characterization is partial and captures only sufficient detail for our purposes.) In step 1, we express the policy goals that authentication has to be fast and also secure.

Step 1:

Policy Goal 1	exec_time $\leq$ b
Policy Goal 2	sec_req $\leq$ t

Goal 2 says that the actually security level provided is not lower than that required. Next, we analyze the relevant system constraints. Let d denote the message transmission time between any two nodes (assuming for convenience that the nodes are all uniformly located throughout the network).

Step 2:

System Constraint 1	exec_time = r $\times$ d + O(nlog (t))
System Constraint 2	2 $\times$ t + 1 $\leq$ n

The two constraints are non-negotiable properties of distributed authentication. The first reflects that the time to complete the protocol is the time of the many rounds of communication with the servers (including the time for a full handshake) plus the computational complexity at the end nodes. The second is that the corruption of up to half of the operational servers can be tolerated [10].

We can see from the requirements that the time limit on protocol execution may conflict with the security level requirement. When t increases, the execution time also increases and may exceed the limit b. To decide the feasibility of constraints, we form the following programming problem in step 3 of our framework.

Step 3:

$$\begin{cases} \texttt{exec\_time} \leq \texttt{b} \\ \texttt{sec\_req} \leq \texttt{t} \\ \hline \texttt{exec\_time} = \texttt{r} \times \texttt{d} + \texttt{O}(\texttt{n}\log(\texttt{t})) \\ 2 \times \texttt{t} + 1 \leq \texttt{n} \end{cases}$$

Given the potential for conflict, a trade-off rule is needed. A common rule is that security takes priority over performance – that is, one is willing to concede performance to maintain

security. This rule can be encoded as a requirement by moving policy goal exec_time $\leq b$ into a (negotiable) system constraint and relaxing it with exec_time $\leq$ b + z, and then adding a new goal to minimize z.

Step 4:

$$\begin{cases} \texttt{minimize z} \\ \texttt{sec\_req} \leq \texttt{t} \\ \hline \texttt{exec\_time} \leq \texttt{b} + \texttt{z} \\ \texttt{exec\_time} = \texttt{r} \times \texttt{d} + \texttt{O(n log(t))} \\ 2 \times \texttt{t} + 1 \leq \texttt{n} \end{cases}$$

It is also clear from the requirements that availability and security may conflict. For example, given a fixed number of servers, requiring a security level of t means that $2t+1$ servers must be operational for the authentication service to be available. However, if some servers have crashed (e.g., because of power failures) or network links have been severed, the total number of available servers is reduced to the extent that security level t cannot be maintained. In this situation, there are two choices. One is to keep the same security level and tolerate unavailability (i.e., the programming problem has no solution); the other is to lower the security level as a means to increase service availability. For certain systems, a new trade-off rule may be that availability takes priority over security – in other words, the security level becomes a negotiable requirement. This rule can be encoded by moving policy goal sec_req $\leq$ t into a (negotiable) system constraint and relaxing it with sec_req $\leq$ t - w, and then adding a new goal to minimize w. We assume that f is the absolute minimum security level.

Step 4 (new):

$$\begin{cases} \texttt{minimize z} \\ \texttt{minimize w} \\ \hline \texttt{f} \leq \texttt{sec\_req} \leq \texttt{t} - \texttt{w} \\ \texttt{exec\_time} \leq \texttt{b} + \texttt{z} \\ \texttt{exec\_time} = \texttt{r} \times \texttt{d} + \texttt{O(n log(t))} \\ 2 \times \texttt{t} + 1 \leq \texttt{n} \end{cases}$$

In addition, if the first rule takes precedence over the second rule, we can encode such priorities as follows. Let M be a large positive number. We can combine the policy goals as follows.

Step 4 (combined):

$$\begin{cases} \texttt{minimize M} \times \texttt{z} + \texttt{w} \\ \hline \texttt{f} \leq \texttt{sec\_req} \leq \texttt{t} - \texttt{w} \\ \texttt{exec\_time} \leq \texttt{b} + \texttt{z} \\ \texttt{exec\_time} = \texttt{r} \times \texttt{d} + \texttt{O(n log(t))} \\ 2 \times \texttt{t} + 1 \leq \texttt{n} \end{cases}$$

In step 5 of the framework, the trade-off rules are implemented in the authentication management system (not discussed here) and the programming problem is solved whenever the working environment changes (e.g., when machines fail).

7 Case Study III: Security and Fault Tolerance

In this case study, again we first motivate the trade-offs and then review background materials. After that, we apply our framework to show how the trade-offs can be handled.

7.1 Motivation

In building fault tolerance services in a distributed system, there are two major approaches, namely, the Primary-Backup (PB) approach (e.g., [1, 4]) and the State-Machine (SM) approach (e.g., [27, 23]). Each approach has its distinctive advantages. To tolerate simple faults such as crash and omission, PB protocols are in general significantly cheaper than SM protocols in terms of numbers of processors, messages, and rounds (which directly affect the service response time). PB protocols are also much simpler than SM protocols, and thus the efforts of debugging or formal verification of PB protocols are also easier. On the other hand, in choosing to run a PB protocol instead of a SM protocol, one risks providing incorrect service functions or values that may cause the overall system to fail when more serious *types* of faults such as arbitrary (Byzantine) faults occur. (Any given system configuration can tolerate only up to a certain *number* of faults. The emphasis here is on the distinction that a PB protocol cannot tolerate Byzantine faults.) Therefore, it is common practice for critical applications to run an SM protocol, possibly using Byzantine agreement [14]. The high cost of running such a protocol is compensated by the knowledge that all possible faults (up to a certain number) are adequately tolerated.

Instead of being forced to make a design choice between the SM or the PB approaches, and thus either incurring a high running cost or risking system failure when unexpected faults occur, system developers can now use a newly proposed approach, called *adaptive fault tolerance* [9]. Given that in many situations Byzantine or other nontrivial faults occur only relatively infrequently, an intelligent adaptive algorithm has been developed, using PB and SM protocols as building blocks, that runs typically at a cost close to that of a PB protocol and switches to a more expensive SM protocol only when complicated faults (faults cannot be tolerated by a PB protocol) occur. This adaptive approach thus has the potential to retain the desirable attributes of both PB and SM approaches. Moreover, the adaptive approach is modular, in that any PB or SM protocol can be used [11].

For noncritical applications, the adaptive fault tolerance approach may be seen as a way to extend PB to cover more complex faults at low additional cost. For critical applications,

the approach may be seen as a way to allow some of the processing resources required for SM to be used for other services when full SM functionality is not needed. For example, when only manifest faults occur, an adaptive algorithm runs in the PB mode and can thus tolerate a maximum number of such benign faults. The adaptation can also be viewed as an adaptation between an optimistic algorithm and a pessimistic one, where the former is the default mode of operation and the latter is invoked only when necessary.

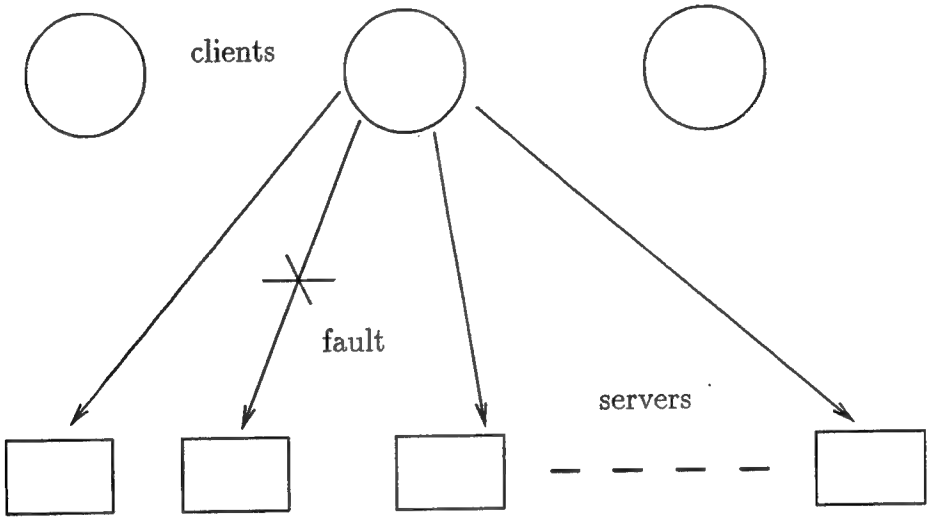


Figure 4: Security and Fault Tolerance

The structure of an adaptive fault-tolerant service is illustrated in Figure 4, using the client-server model. There, a distributed system service is provided by n servers, which usually run in the primary-backup mode and switch to the state-machine mode only when necessary. The system objective is to utilize adaptivity during run time so that sufficient security (i.e., the ability to tolerate arbitrary faults) is maintained and service cost is minimized.

7.2 Background on Adaptive Fault Tolerance

The environment we assume is the following. Clients send their requests to the servers who process the requests and respond, all by exchanging messages. For simplicity, we assume that the communication channel between a client and any server is reliable and FIFO, and we aim to tolerate faulty servers but not faulty clients. We also assume that the servers are deterministic – because in the state-machine approach it is usually undesirable to allow nondeterministic behaviors in the (correct) servers. Moreover, we assume that the system is synchronous, and thus we can use a model of computation based on rounds. The reason for

this limitation is that it is impossible to guarantee both safety and liveness in asynchronous systems [3, p.19].

Following the literature, we classify faults into three categories [15]:

- Manifest fault – one that produces detectably missing values (e.g., crash and omission faults) or that produces a value that all nonfaulty recipients can detect as bad (e.g., it fails checksum or format or typing tests).
- Symmetric fault – one that delivers the same wrong value to every nonfaulty receiver.
- Asymmetric fault – an arbitrary fault with no constraints, also known as Byzantine fault.

Briefly, in Primary-Backup (PB) approach, one and at most one server is designated as the primary at any time. A client sends a request to the primary, who processes it and then broadcasts the necessary state change to all backup servers. In a nonblocking PB protocol, the primary server responds to the client before receiving acknowledgments to its broadcast whereas in a blocking protocol, the primary blocks until all backups have acknowledged or after a timeout period. The schema for a server consists of three modules for: (1) deciding whether it is a primary or a backup, (2) processing requests, and (3) fault detection and recovery [3, p.56]. It is apparent that the PB approach can tolerate only manifest faults. For example, an incorrect primary can broadcast an incorrect state change and backup servers cannot detect this fact because they do not know the client's service request.

In a state-machine (SM) protocol, a client broadcasts its request to all servers, and then takes a vote on the responses it receives. Therefore, the client will decide on the correct response if a majority of the servers are nonfaulty. For correctness, all nonfaulty servers must process requests (possibly from multiple clients) in the same order. This requirement is called replica coordination [23] and is not necessary in a PB protocol. Satisfying this coordination requirement is quite expensive – for example, a Byzantine agreement protocol is a typical solution. With this heavy cost in resources and performance, the SM approach gains the ability to tolerate symmetric as well as asymmetric faults, in addition to manifest faults.

Our algorithm that adapts between tolerating manifest faults and asymmetric faults is given below in Table 2. A more detailed analysis is in [11].

Round 0. *Client*: Broadcast request r to all servers.
Primary: Wait for request from client.
Backup: Wait for request from client.

Round 1. *Client*: Wait for response from primary.
Primary: Broadcast (r, a(r), s-c) to all backups. Respond a(r) to client.
Backup: Queue r. Wait for message from primary.

Round 2. *Client*: Wait for a(r) or error report from backup.
Primary: Wait for error report from backup.
Backup: Verify the correctness of a(r). If correct, respond a(r) to client.
If error, broadcast ERROR to client and all servers,
and start BA protocol (to agree on which client request to process).
Wait for error report from other backups.

Round 3. *Client*: If receive an error report,
wait for the BA protocol to complete; then vote on the responses.
If no error is reported but primary's a(r) is not the majority vote
of the a(r)'s, broadcast to all servers to initiate BA protocol.
Otherwise, accept a(r).
Primary: If receive an error report, switch to the BA protocol, process request,
respond to client, and return to **Round 1**. Otherwise, go to **Round 4**.
Backup: If receive an error report, switch to or continue with the BA protocol,
process request, respond to client, and return to **Round 1**.
Otherwise, go to **Round 4**.

Round 4. *Primary*: Wait to see if client report error. If error, switch to BA protocol.
Otherwise, return to **Round 1**.
Backup: Wait to see if client report error. If error, switch to BA protocol.
Otherwise, return to **Round 1**.

Table 2: Byzantine-On-Demand Algorithm

7.3 Applying the Framework

To apply our framework, we use the following notation.

n – number of servers
r – number of rounds
d – message delivery time
t – security level (number of faults)
b – service response requirement

We can describe the policy goals of the adaptive fault-tolerant service as follows.

Step 1:

Policy Goal 1	exec_time $\leq$ b
Policy Goal 2	sec_req $\leq$ t

Next, we can characterize the system constraints as follows.

Step 2:

System Constraint 1	exec_time = r $\times$ d
System Constraint 2	3 $\times$ t + 1 $\leq$ n
System Constraint 3	t + 1 $\leq$ r

These constraints are the properties of Byzantine agreement [23], and are non-negotiable. The performance (service response) requirement and the number of faults that may be tolerated can conflict, as we can find out in step 3.

Step 3:

$$\begin{cases} \texttt{exec\_time} \leq \texttt{b} \\ \texttt{sec\_req} \leq \texttt{t} \\ \hline \texttt{exec\_time} = \texttt{r} \times \texttt{d} \\ 3 \times \texttt{t} + 1 \leq \texttt{n} \\ \texttt{t} + 1 \leq \texttt{r} \end{cases}$$

This linear programming problem (assuming b being a constant) may not have a solution because a high level of security demands a large t, which will increase r and hence the exec_time.

Tradeoff rules can be similarly developed as before. For example, we can relax the real-time requirement by introducing variable z, moving policy goal $\texttt{exec\_time} \leq \texttt{b}$ to be a system constraint $\texttt{exec\_time} \leq \texttt{b} + \texttt{z}$, and adding a new goal `minimize z`, as shown below.

Step 4:

$$\begin{cases} \texttt{minimize z} \\ \texttt{sec\_req} \leq \texttt{t} \\ \hline \texttt{exec\_time} \leq \texttt{b} + \texttt{z} \\ \texttt{exec\_time} = \texttt{r} \times \texttt{d} \\ 3 \times \texttt{t} + 1 \leq \texttt{n} \\ \texttt{t} + 1 \leq \texttt{r} \end{cases}$$

In step 5 of the framework, the trade-off rules are implemented in the fault tolerance management system and the linear programming problem is solved whenever the working environment changes (e.g., when machines fail).

8 Conclusions and Future Work

Computer systems implementing mission-critical applications typically must be adaptive to events and changes in the operating environment; thus, a system often must make dynamic trade-offs in order to attain its various and often conflicting objectives. Although there have been isolated studies on such system trade-offs, such as Secure Alpha [2], there is a need for a general framework to handle potentially conflicting critical system properties.

In this final report, we have proposed such a general framework within which multiple critical system properties can be specified, multiple requirements can be jointly considered, and conflicts can be resolved. Our framework is driven by examples, which we also use, with specific conflict resolution rules, to illustrate how the framework can be useful. Our framework consists of five major steps:

1. Establishing policy goals: understand and write down system requirements in precise, mathematical and logical terms.
2. Establishing system constraints: understand and write down the target system characteristics that are directly or indirectly related to and can affect the realization of the policy goals.
3. Deciding feasibility of goals: use mathematical methods to seek a feasible solution of policy goals under the given system constraints.
4. Providing directions for trade-offs: specify rules and priorities of trade-offs between potentially conflicting policy goals and system configuration.

5. Realizing trade-offs: use mathematical methods to seek a feasible and perhaps best solution of policy goals under the given system constraints, following the trade-off rules.

Our approach has the following advantages. First, the framework is unified in that a wide range of critical system properties, both quantitative and qualitative, can be considered in the same way. Thus, it is easy to add new (types of) requirements, and in addition, some common trade-off rules also can be encoded as requirements. Second, there exist systematic methods (e.g., Gaussian elimination and Shostak's method [25, 26]) and basic tools (e.g., the theorem-proving system PVS [20]) that can be used to detect conflicts and to evaluate trade-offs. Some trade-off rules can be encoded as optimization problems that can be solved with standard and mature methods in linear (and integer) programming [6, 12, 21, 22].

The three detailed case studies of Secure Alpha [2], distributed authentication [10], and adaptive fault tolerance [11], clearly demonstrate how our framework provides a common underpinning and uniform treatment to various trade-offs between security, real-time requirement, fault tolerance, and so on. We thus can conclude that the framework is useful and effective.

There are a number of possible directions for future work. One is to work on complex system examples, perhaps applying the framework to existing systems in order to gain more insight into the problem area and to test the scalability of the approach. For example, we may encounter trade-off rules that are difficult to express or handle within the current framework. These examples may suggest ways to extend the framework – such as developing heuristics and fast algorithms for locating conflicts.

Another direction might be to provide a semantic foundation for the framework. For example, the description and translation of requirements from the system description into our framework is informal as described, and the framework currently does not provide evidence that the translation accurately reflects the original system description. Thus, it is useful to have a semantics on which a more formal translation can be based.

A third direction might be to construct a tool for system analysis and simulation, so that potential conflicts and trade-offs can be discovered at design time, trade-off strategies can be tested by simulation or analysis, and important design decisions can be made with these conflicts in consideration.

Finally, it might be attractive to build a set of reusable system services, including application programming interfaces. Such services could act as "middleware" for implementing trade-offs and realizing different security policies in a heterogeneous environment.

Acknowledgments

In writing this report, we made extensive use of materials in some previously published papers, especially [2, 10, 11]. Emilie Siarkiewicz of U.S. Air Force Rome Laboratory and our colleagues at SRI provided useful discussions and feedbacks on earlier drafts of this report.

References

[1] P.A. Alsberg and J.D. Day. A Principle for Resilient Sharing of Distributed Resources. In *Proceedings of the 2nd International Conference on Software Engineering*, pages 627–644, October 1976.

[2] P.K. Boucher, R.K. Clark, I.B. Greenberg, E.D. Jensen, and D.M. Wells. Toward a Multilevel-Secure, Best-Effort Real-Time Scheduler. In *Proceedings of the 4th IFIP Working Conference on Dependable Computing for Critical Applications*, pages 33–45, San Diego, California, January 1994.

[3] N. Budhiraja. *The Primary-Backup Approach: Lower and Upper Bounds.* Ph.D. dissertation, Cornell University, Ithaca, New York, June 1993. Available as Technical Report 93-1353.

[4] N. Budhiraja, K. Marzullo, F.B. Schneider, and S. Toueg. Primary-Backup Protocols: Lower Bounds and Optimal Implementations. In *Proceedings of the 3rd IFIP Working Conference on Dependable Computing for Critical Applications*, pages 187–196, Sicily, Italy, September 1992.

[5] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Process of Faults. In *Proceedings of the 26th IEEE Symposium on the Foundations of Computer Science*, pages 383–395, October 1985.

[6] G.B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, New Jersey, 1963.

[7] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–65, November 1976.

[8] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Co., New York, 1979. Paperback edition 1991.

[9] J. Goldberg, I. Greenberg, and T.F. Lawrence. Adaptive Fault Tolerance. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 127–132, Princeton, New Jersey, October 1993.

[10] L. Gong. Increasing Availability and Security of an Authentication Service. *IEEE Journal on Selected Areas in Communications*, 11(5):657–662, June 1993.

[11] L. Gong and J. Goldberg. Implementing Adaptive Fault-Tolerant Services for Hybrid Faults. Technical Report SRI-CSL-94-04, Computer Science Laboratory, SRI International, Menlo Park, California, March 1994. Revised September 30, 1994.

[12] T.C. Hu. *Combinatorial Algorithms.* Addison-Wesley, Menlo Park, California, 1982.

[13] S.C. Kothari. Generalized Linear Threshold Scheme. In *Advances in Cryptology: Proceedings of Crypto '84*, volume 196 of *Lecture Notes in Computer Science*, pages 231–241. Springer-Verlag, New York, 1984.

[14] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[15] P. Lincoln and J. Rushby. A Formally Verified Algorithm for Interactive Consistency Under a Hybrid Fault Model. In *Proceedings of the 23rd Fault-Tolerant Computing Symposium*, pages 402–411, Toulouse, France, June 1993.

[16] R.C. Merkle. A Fast Software One Way Hash Function. *Journal of Cryptology*, 3(1):43–58, 1990.

[17] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[18] R.M. Needham and M.D. Schroeder. Authentication Revisited. *ACM Operating Systems Review*, 21(1):7, January 1987.

[19] D. Otway and O. Rees. Efficient and Timely Mutual Authentication. *ACM Operating Systems Review*, 21(1):8–10, January 1987.

[20] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In *11th Conference on Automated Deduction*, volume 607 of *Lecture Notes on Artificial Intelligence*, pages 748–752. Springer-Verlag, June 1992.

[21] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, New Jersey, 1982.

[22] E.M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, New Jersey, 1977.

[23] F.B. Schneider. Implementing Fault-Tolerant Services Using the State-Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[24] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[25] R. Shostak. An Efficient Decision Procedure for Arithmetic with Function Symbols. Technical Report CSL-65, SRI International, Computer Science Laboratory, Menlo Park, California, September 1977.

[26] R. Shostak. Deciding Linear Inequalities by Computing Loop Residues. *Journal of the ACM*, 28(4):769–779, October 1981.

[27] J.H. Wensley, L. Lamport, J. Goldberg, M.W. Green, K.N. Levitt, P.M. Melliar-Smith, R.E. Shostak, and C.B. Weinstock. SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control. *Proceedings of the IEEE*, 66(10):1240–1255, October 1978.

Rome Laboratory

Customer Satisfaction Survey

RL-TR-_____

Please complete this survey, and mail to RL/IMPS, 26 Electronic Pky, Griffiss AFB NY 13441-4514. Your assessment and feedback regarding this technical report will allow Rome Laboratory to have a vehicle to continuously improve our methods of research, publication, and customer satisfaction. Your assistance is greatly appreciated.
Thank You

_____

_____

Organization Name:_____(Optional)

Organization POC: _____(Optional)

Address:_____

1. On a scale of 1 to 5 how would you rate the technology developed under this research?

5-Extremely Useful 1-Not Useful/Wasteful

Rating_____

Please use the space below to comment on your rating. Please suggest improvements. Use the back of this sheet if necessary.

2. Do any specific areas of the report stand out as exceptional?

Yes___ No_____

If yes, please identify the area(s), and comment on what aspects make them "stand out."

3. Do any specific areas of the report stand out as inferior?

Yes___ No___

If yes, please identify the area(s), and comment on what aspects make them "stand out."

4. Please utilize the space below to comment on any other aspects of the report. Comments on both technical content and reporting format are desired.

*U.S. GOVERNMENT PRINTING OFFICE: 1995-610-126-20013

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.